One Requirement, One Statement

The Problem:

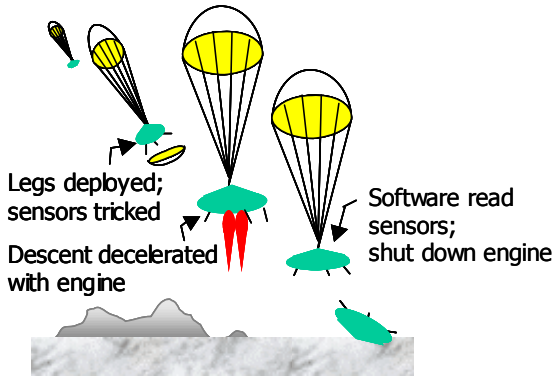
Contact to an interplanetary probe was lost shortly before touchdown. Most likely, a spurious signal prematurely shut down the descent engines, causing the spacecraft to crash.

The Cause:

As the lander parachuted, it deployed three legs, each with a sensor designed to command the engine off upon touchdown lest the lander overturn.

Leg deployment was known to trigger false signals in the sensors, and the systems spec indicated that the engine should be protected from such transients.

Unfortunately, the protection clause was buried in another requirement, as follows: "The sensors shall… *However, the use of the sensor data shall not begin until….*" This "*However...*" phrase was not picked up by the software team or by other subsystems, and was not specifically tested at the system level.



Leg deployment triggered a spurious reading that should have been ignored. When the touchdown sensing logic was enabled, the software read the false data and terminated the engine, causing the crash.

The software walkthrough and integration/test did not detect this problem (logic flow diagrams could have helped). What's more, a leg-deployment test failed to detect the fault because the sensors were improperly wired at first. A rerun of the deployment test, which might have caught the error, was not performed after rewiring.

Lessons Learned:

- Do not lump several requirements together—write them out separately so that each can be tracked individually. Negative statements (e.g., "Sampling shall *not* begin until…") may cause misunderstanding and should be avoided.
- Systems engineers must take ownership of requirements and partition them to the appropriate subsystem. Whether or not a requirement is the software's responsibility, for example, should not be left to the discretion of the software team.
- Systems engineering must ensure thorough end-to-end failure mode testing.
- The software review process should emphasize logic flow. Tests should exercise every requirement to see if there are conditions that could cause the software to fail.
- Test planning needs to consider transients or spurious signals.
- When important tests are aborted or are known to be flawed, they must be rerun after the errors are fixed. Repeat the test if any software or hardware involved are changed.

For comments on the Aerospace Lessons Learned Program, including background specifics, call Paul Cheng at (310) 336-8222.